

# Fast and Lazy Build of Acceleration Structures from Scene Hierarchies

Warren Hunt\*

William R. Mark†

Don Fussell‡

Department of Computer Sciences  
The University of Texas at Austin

## ABSTRACT

In this paper we show how to use structural information about a scene such as is contained in a scene graph to build SAH-based acceleration structures more efficiently. We provide a general method for doing so together with asymptotic analyses for both standard and lazy variants of our method. In particular, we show bounds of  $O(n)$  for full k-d tree builds over  $n$  primitives and  $O(v + \log n)$  for lazy k-d tree builds over  $v$  visible primitives. We provide experimental results showing that these asymptotic properties translate into real-world speedups. In fact, without a method like ours, it is impossible to achieve better than  $O(n)$  for even the first split of a lazy build. We also show that under certain (realistic) assumptions on the scene structure, our method produces provably good acceleration structures. Finally, we provide experimental results demonstrating that our acceleration structures are of nearly indistinguishable quality to those produced with a full SAH build.

**Keywords:** k-d tree, scene graph, ray tracing, acceleration structure

**Index Terms:** I.3.7 [Computer Graphics]: Three-dimensional graphics and realism—Raytracing;

## 1 INTRODUCTION

Recent technological and algorithmic improvements have made interactive and even real-time ray tracing a reality on modern processors. Perhaps the most significant challenge in expanding the capabilities of modern real-time ray tracers is to handle dynamic scenes, in which objects are moving with respect to each other and possibly deforming as well. For such scenes, ray tracers cannot rely on pre-computed static acceleration structures that have been optimized for fast traversal; any acceleration structure used must be able to adapt to changing scene characteristics in real time. Until recently, this has resulted in the use of simple acceleration structures such as regular grids, median bounding-volume hierarchies (BVHs), or poorly-optimized k-d trees, that can be updated or rebuilt quickly each frame [12, 15, 8, 13]. However, recent work has made it possible to construct much more highly-optimized k-d tree accelerators using the surface-area heuristic (SAH) efficiently enough so that they can be rebuilt on a frame-by-frame basis in real time [7, 11]. These highly-optimized structures are more effective at accelerating the rendering process than other techniques. If they can be rebuilt quickly, they can provide overall faster real time performance than was previously possible.

Most existing SAH-based tree construction algorithms have assumed that the input data is essentially a “soup” of polygons or axis-aligned bounding boxes (AABBs) [7, 11, 14]. That is, no global scene structure is assumed, or taken advantage of, in building the tree. However, real scenes have a great deal of structure.

This structure is generally encoded as a scene graph representing the natural boundaries between, and relationships among, the objects in a scene. A scene graph is a hierarchical object. Rigid body motions in the scene are represented by transformations between local coordinate frames associated with the objects represented in the scene graph. Natural clusters of geometry in the scene are found within objects. Thus, the information available in scene graphs is quite pertinent to the heuristics used for ray-tracing acceleration. One simple way in which a scene graph can be used to build an acceleration structure is to create a BVH that mirrors the hierarchy in the scene graph, where each object (or instance) in the scene graph contributes its bounding volume to the BVH.

While some batch ray tracers have exploited scene graph information, for example to enable lazy creation of acceleration structures, the Razor interactive ray tracer [2] is the first to our knowledge to exploit such information in an interactive or real-time system. Razor incorporates a fast SAH-based k-d tree build scheme based on an input scene graph and using the scan-based surface-area metric (SAM) approximation scheme of [7]. Use of an input scene graph also enables Razor to use a lazy build scheme to further accelerate the k-d tree builder.

In this paper, we introduce a scheme for building SAH-based acceleration structures from input hierarchies that extends the k-d tree build algorithm of Razor to a more general class of acceleration structures. We show that our scheme is asymptotically fast and that in practice it produces acceleration structures as good as those produced from scratch, as long as the input scene structure is as well-behaved as most common scene graphs. In particular, for input that is in some sense “presorted”, we show that our method performs a full SAH-based hierarchy build over  $n$  primitives in  $O(n)$  time that is in some provable sense high-quality. This can be compared to the known bound of  $n \log n$  for tree construction using the SAH with no assumptions about scene structure [14].

We also show that, given an input hierarchy, we can perform a lazy k-d tree build over  $v$  visible primitives in a scene in  $O(v + \log n)$ . Given that  $v \ll n$ , this result is substantially better than either a lazy build without scene structure or a non-lazy build using scene structure, which are  $O(n)$  or worse. We then show how the fast scan-based SAM approximation scheme of [7] can be used to improve the performance of our build methods by a constant factor.

Finally, we use the k-d tree builder in Razor to demonstrate experimentally that our asymptotic efficiencies translate into real-world performance gains and that the trees produced are as good as those produced without exploiting scene structure. For example, by using a lazy strategy to build from a scene graph, we can build good acceleration structures an order of magnitude faster than by building them from scratch.

We conclude from these results that exploiting the structural information provided in scene graphs is key to achieving fast build speeds along with high rendering efficiency for ray-tracing acceleration structures.

\*email: whunt@cs.utexas.edu

†email: billmark@cs.utexas.edu

‡email: fussell@cs.utexas.edu

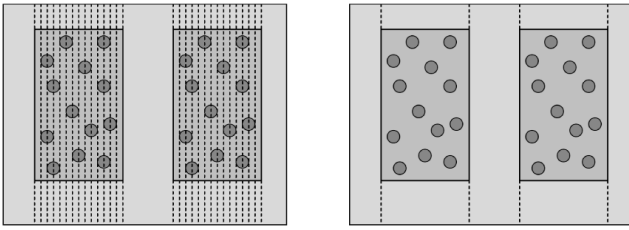


Figure 1: The use of hierarchy can greatly reduce the number of split candidates provided to a builder. If the hierarchy fits the geometry well, then the provided splits are often among the best anyway.

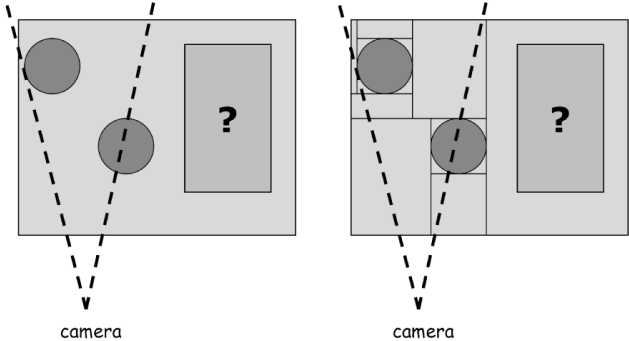


Figure 2: Using hierarchy in conjunction with lazy build can allow a builder to completely ignore large regions of space if no rays enter that space.

## 2 BACKGROUND

### 2.1 Building acceleration structures for dynamic scenes

In batch ray tracing systems, the construction of the acceleration structure was often considered to be a “free” pre-processing step. However, as interactive ray tracing has become practical, it has become important to consider fast techniques for building and updating acceleration structures.

For scenes containing deformable motion, three broad strategies have recently emerged [16]. The first is to choose an acceleration structure such as a regular grid that is especially simple to build [15]. The second is to use high-performance algorithms to build a high-quality acceleration structure such as an SAH-based kd-tree [7, 11] or BVH [13]. The third is to incrementally update the acceleration structure each frame [8, 13, 17]. We focus on the second strategy in this paper. Our analysis assumes a pre-existing hierarchy as the starting point for a build. This can come from an input scene graph, or it would be possible to use an updated version of the previous frame’s acceleration structure as this pre-existing hierarchy. In the latter case, the distinction between rebuilding and updating the acceleration structure would become blurry.

### 2.2 Building k-d trees using the SAH

The traditional algorithm for building a k-d tree (summarized in [10]) is a greedy, top-down algorithm using the SAH to evaluate split candidates. The decision as to where and whether to split a voxel is based on a recursively-defined cost function, the surface-area metric (SAM) [3, 9, 6]:

$$C_{SAM}(x) = C_I + C_L(x) \frac{SA_L(v,x)}{SA(v)} + C_R(x) \frac{SA_R(v,x)}{SA(v)},$$

where  $C_I$  is the (constant) cost of traversing the node representing the voxel being considered for splitting, whose surface area is given

by  $SA(v)$ .  $C_L(x)$  is the cost of the left child and  $SA_L(v,x)$  its surface area, given a split at candidate position  $x$ . Likewise,  $C_R(x)$  is the cost of the right child and  $SA_R(v,x)$  its surface area given the same split candidate. The ratios  $\frac{SA_L(v,x)}{SA(v)}$  and  $\frac{SA_R(v,x)}{SA(v)}$  can be interpreted as the conditional probabilities that a ray will intersect the left and right child respectively given that it intersects the current node.

For a given voxel, the split with lowest cost across all split candidates is compared to the cost of not splitting the voxel. If the cost of splitting is less than the cost of not splitting, the voxel is split into two children. Otherwise, a leaf node is created in the tree. Splits chosen using this cost function produce trees that have proven to be very effective for fast ray tracing.

The candidate split positions are defined by the geometry that is being partitioned. We assume that each piece of geometry is contained within an axis aligned bounding box (AABB). The upper and lower bounds of each AABB make up the list of candidates. In this case  $C_L(x)$  and  $C_R(x)$  are equal to the number of AABBs that overlap the left and right children respectively. The difficult part in evaluating the cost function for a given candidate  $x$  is in the evaluation of  $C_L(x)$  and  $C_R(x)$ . When evaluating all split candidates, this task is simplified greatly if all of the splits are sorted with respect to their position (in one dimension), because a linear scan can keep track of both the number of AABBs to the left of a given split position and the number of AABBs that overlap it. This information is enough to derive  $C_L$  and  $C_R$ . Traditionally, the split candidates are sorted each time a new split is required, leading to an  $n^2 \log n$  tree-building algorithm. By doing all the sorting up front [1, 14], a better bound of  $n \log n$  can be achieved. With even modest amounts of geometry, any sort is still expensive if it is to be computed each frame. It has recently been shown that significant improvements in performance can be made by approximating the SAH value instead of computing it exactly [7, 11], although the asymptotic bound for these methods remains  $n \log n$ .

These methods can be generalized easily to construction of a BVH by simply grouping the objects in the scene according to the positions of their centroids with respect to the split planes.

## 3 USING HIERARCHY TO ACCELERATE ACCELERATION STRUCTURE BUILD

In all of the work cited above, it has been assumed that the acceleration structure is being built “from scratch”, i.e. that no global structural properties of the scene are known when building the structure.

Good acceleration structures perform two roles in accelerating ray tracing. First, they help eliminate wasted work testing for intersections with objects that are not near a ray, and second, they allow the intersection tests that are done to be ordered along that ray. This ordering essentially amounts to sorting the scene geometry, and this sorting requirement is the source of the  $\Theta(n \log n)$  complexity of building the acceleration structure. However, in practice, a great deal of structural information about scene geometry is usually known at the time the acceleration structure is built. Such information could be obtained, for example, from a scene graph, or in a real-time context it could be obtained from a previous acceleration structure built to accelerate an earlier frame. When this structural information is sufficient to amount to a “pre-sort” of the scene geometry, using it can allow a new acceleration structure to be built in linear time rather than  $O(n \log n)$ .

We now sketch our approach for building an acceleration structure over  $n$  primitives in  $O(n)$  time. We achieve linear complexity by doing constant work to determine each split plane during acceleration structure build. This is done by only examining a constant-size subset of scene objects when determining each SAH cut plane. Because sorting a constant-size set takes constant time, we can guarantee constant work per node. Our algorithm obtains this constant number of objects via careful refinement of the provided

scene-graph BVH. Our approach is also guaranteed (under reasonable assumptions) to produce a “good” acceleration structure. The primary assumption required is that each of these constant-size subset of objects has a normal enough spatial distribution that a “good” acceleration structure exists for it. In this case, we can show that our algorithm will produce a structure of quality comparable to the quality of a structure produced using a full SAH build. Thus, with an appropriate input structure, our method will produce an acceleration structure guaranteed to be high quality and do so in linear time.

In the remainder of this paper we will formalize our algorithm and the intuitions we have provided. First we will deal with the issue of what exactly is meant by a “good” acceleration structure using the SAH. Then we will define more precisely the structural requirements on the input that will allow us to produce good acceleration structures on output. We will show a bit more formally why this process takes linear time. We will then examine various strategies for enhancing this basic scheme to provide better structure quality and safeguard against failed assumptions. Finally, we provide experimental confirmation of our more formal analyses.

#### 4 HIGH QUALITY SAH-BASED ACCELERATION STRUCTURES

The SAM has long been the most commonly used basis for heuristics for building high quality acceleration structures. This is because the SAM captures the expected cost of processing a ray through the structure, given that the ray is drawn from a uniform distribution over ray space. Thus, the SAM is an absolute quality metric; minimizing it means minimizing the expected cost of processing a ray. The SAH, in this context, is a heuristic in the sense that it is not guaranteed to produce a structure that globally minimizes the expected cost of processing a ray for a given scene geometry. This global minimum is generally not known as it is intractable to compute. So, given this absolute metric, it is very easy to determine which of two trees is better, but lacking the global minimum value, it is still difficult to say if a given tree is “good”. When comparing trees produced by our algorithm to trees produced by the standard SAH algorithm, we require some notion that our trees are “as good” as the standard trees. Since they’re not likely to be better, we need to define “good”.

When a ray reaches a leaf of an acceleration structure, it tests for intersections with all of the geometry in that leaf. To limit the work done at a leaf, we would like leaves of a good acceleration structure to be at most constant  $c$  size. Not only would a good acceleration structure have limited-size leaves, but the fanout of any node in the hierarchy should likewise be bounded by a constant, call that value  $F$ . For the same reason as in the case of the leaves, a ray tracer has to do more work at high fanout interior nodes. Additionally, in a good structure the bounding volumes of the children of any given node are at least some fraction smaller than their parent, meaning that we have limited overlap of the geometry at each level. For each child, this implies that the conditional probability of that child being intersected by a given ray that intersects its parent is bounded from above, thus helping to minimize the SAM. Not all scene geometry can guarantee this property, but most does in practice.

What kind of SAM cost will such a “good” structure have? The cost of each leaf node in the acceleration structure is at most  $c$ . Let  $p$  be the average conditional probability of intersecting a child in the structure. The cost of the nodes with children that are leaves is  $c(Fp)$ . Similarly, the cost of nodes with grandchildren that are leaves is  $c(Fp)(Fp)$ . By induction, it is easy to show that the SAM cost of the root node is  $C_{SAM} = c(Fp)^{\log_F n} = cn^{1+\log_F p}$ . Since  $p < 1$ , the log part is negative and the order is sub-linear. It is unlikely that the average conditional probability  $p$  is the same for all sets of children, and thus this derivation is only accurate in an ideal case. However, given that we know how to evaluate  $C_{SAM}$  directly, we can invert this formula for SAM cost to obtain an independently

useful “average” value for  $p$  across an entire tree. We will refer to this “average”  $p$  by  $q : q = (\frac{C_{SAM}}{cn})^{\frac{1}{\log_F n}}$ . This new quantity is an abstract measure of tree quality that is independent of  $n$ . This is specifically different than the  $C_{SAM}$  cost, which is dependent on  $n$ . Note that lower values of  $q$  represent higher-quality trees. We will use  $q$  to compare trees with very different sizes and costs.

Rewriting our above formula to include our new variable, a tree of quality  $q$  has SAM cost:  $C_{SAM} = cn^{1+\log_F q}$ . It is very important to understand that this is a quality metric, not a cost metric. The SAM is a cost metric, which will tell you how much work traversing a ray through the structure costs. This quality metric will tell you how good a tree is, independent of its size. Given a quality, you will need to provide a size ( $n$ ) and a fanout ( $F$ ) to derive its cost. This quality metric, on its own, provides an abstract value of “goodness” on a tree without incorporating the tree’s size or fanout.

Tree quality  $q$  has a valid range  $0 \leq q \leq c^{\frac{-1}{\log_F n}} \leq 1$  due to the fact that the SAM cost function on a tree produced by the SAH is bounded between 1 and  $n$ . That is, the SAH cannot produce a tree of negative cost: it will abort and produce a node of cost  $n$  in preference to a tree of cost  $> n$ . The higher upper bound of 1 is easy to check by noting that  $C_{SAM} = cn^{1+\log_F 1} = cn$ , which is  $> n$  for  $c > 1$ . We will define a high-quality (“good”) tree to be one such that its quality  $q$  is  $\leq Q$  for a provided quality threshold  $Q$ .

It should be noted that our notion of “good” does not require that good trees have low fanout. For example, take a binary SAH hierarchy with cost  $2n^{1+\log_2 q}$ ,  $q \leq Q$  and collapse it into an  $h$ -ary hierarchy, it obtains the property that it has cost  $hn^{1+\log_h q_{new}}$ ,  $q_{new} \leq q \leq Q$ . In an  $h$ -ary hierarchy, each leaf costs  $h$ . One level up from the leaves costs  $h(hp')$ , where the average  $p'$  for all children must be no greater than the average  $p$  for those children in the binary tree. This is due to the fact that collapsing interior nodes cannot decrease, and may increase, the size of parents relative to their children, thus likely decreasing the probability of intersecting any given child. By the same induction used in the derivation of the cost formula, we can show that the new  $\log_h n$  depth tree has cost  $hn^{1+\log_h q_{new}}$ . Thus although the trees have different fanout and very different SAM cost, they are related closely in quality. The fact that the  $h$ -ary tree is actually of higher quality than the binary tree may seem puzzling at first. This does not mean that its SAM cost is less, the formula for  $C_{SAM}$  clearly shows that higher fanout will increase the cost of a tree, other things being equal. Intuitively, the high quality of the  $h$ -ary tree comes from the fact that it was distilled from a binary tree with substantially more information about the distribution of the geometry in the scene. The proof of our method’s quality guarantee will show that it is possible to transform a tree in the opposite direction (reduce fanout) in linear time while maintaining high quality, given reasonable assumptions about the input high-fanout tree.

#### 5 BUILDING ACCELERATION STRUCTURES FROM INPUT HIERARCHIES

In this section, we describe our algorithm for constructing an SAH-based acceleration structure from an existing hierarchy. We show that our algorithm will produce a good SAH structure according to the criterion derived above, given appropriate structural assumptions on the input. We also show that under reasonable assumptions our algorithm works in  $O(n)$  time on a scene with  $n$  primitives.

Our algorithm is based on a traditional SAH-acceleration-structure-build algorithm. At each step a split plane is chosen, and present geometry is sifted into two bins based on its position relative to the split position. How this geometry is sifted can be different depending on whether we are constructing a k-d tree or a BVH. We can use groups of objects represented by higher nodes in the scene graph BVH in place of individual objects in order to reduce the amount of geometry considered when choosing such a split plane.

We pick a threshold  $a$  such that we only consider  $O(a)$  objects when determining a split plane. In practice, this number can be any number  $> 1$ . Smaller numbers result in faster build times but increase the dependence on the quality of the input structure. If you have a very high-quality input hierarchy and are greatly concerned with performance, it can be as low as 10. 50 seems sufficient in practice for most cases, but in the Razor system the build takes such a small portion of the overall rendering time, we use 500 to ensure high-quality output trees. Use of 500 objects for determining split planes produces trees of virtually indistinguishable quality to trees produced by the full SAH, in practice.

Note that we want the number of objects selected to be at least  $a$  whenever possible so that we will have a sufficient number of split candidates from which to choose a good split plane. We can collect these objects by refining the input hierarchy nodes one at a time in a top-down order until we have at least  $a$  nodes. In some sense it doesn't matter how we choose the nodes to refine, but using a FIFO queue (breadth first) works well.<sup>1</sup> If we have a binary input hierarchy, we can always obtain exactly  $a$  objects, otherwise we will obtain at most  $a + F_I - 1 = O(a + F_I)$  objects, where  $F_I$  is the maximum fanout of the input hierarchy. The one exception to this is if there are simply not enough objects, even once all of the nodes have been refined. However any  $x : x < a$  is still  $O(a + F_I)$ .

Split planes are chosen according to the SAM as is normally done in using the SAH, where a split candidate is the lower bounding plane of a node and the cost of a node in the SAM is weighted by the number of geometric primitives in that node's subtree. If the SAH cannot find a split over our  $x : x < a + F_I$  objects, then the algorithm is terminated and a leaf node is constructed.

By reducing the amount of geometry considered to a constant, we can bound the work done to determine each split plane to a constant, a significant improvement over previous approaches. It should again be noted that an acceleration structure created with this approach will not exactly match an acceleration structure created considering all split planes, as that different geometry will lead to different decisions made by the SAH builder. It should also be noticed that changing the input geometry can change the point at which the algorithm chooses to terminate the build process and produce a leaf. However, given a reasonable input hierarchy, the algorithm will produce good quality results, as we will show next.

## 5.1 Quality analysis of our algorithm

A local strategy of achieving linear-time build performance by choosing constant-size subsets of the geometry in a scene at each step is not guaranteed to produce a high-quality acceleration structure. Indeed, we rely on the properties of the input scene description to make this possible. Given a scene, we would like to show that our algorithm produces an acceleration structure, from a hierarchy, of comparable quality to the structure produced by the SAH on the same geometry without the use of hierarchy. That is, for an input quality threshold  $Q : Q \leq 1$ , we would like to show that if the SAH build without hierarchy would produce a tree of quality  $(2n^{1+\log_2 q})$  for some  $q : q \leq Q$  then our algorithm will also produce a tree of quality  $2n^{1+\log_2 q'}$  for some  $q' : q' \leq Q$ .

In order to prove this property, we rely on the following assumptions about the input hierarchy and scene geometry.

1. The hierarchy provided must have SAM quality of  $(F_I n^{1+\log_2 q})$ ,  $q \leq Q$  for the provided  $Q$ . This assumption asserts that the input hierarchy is a "rough sort" of the geometry. Without it, it is impossible to construct a high-quality tree in linear time. It should be noted that scene graphs commonly have this property in practice, as do acceleration structures constructed using the SAH for previous frames.

<sup>1</sup>Perhaps some best-first approach would provide a better tree.

2. A  $(cx^{1+\log_2 q})$ ,  $q < Q$  quality acceleration structure exists for every  $x : x < a + F_I$  collection of refined objects from each node (with average cost  $c$ ) in the hierarchy and that the traditional SAH build algorithm will find one such structure. This assumption asserts that all  $x : x < a + F_I$  size collections of objects acquired by refining the scene-graph BVH are well behaved enough that some high-quality structure can be constructed over them. This assumption prevents the occurrence of bad geometry. Such a structure will exist for all selections of geometry that can be partitioned reasonably, (such an assumption is required for all acceleration structures to work well) and holds true for most geometry.

*Observation 1:* If the full SAH will produce a binary tree of cost  $2n^{1+\log_2 q}$  for some  $q : q \leq Q$  given quality threshold  $Q$ , then our algorithm will also produce a  $2n^{1+\log_2 q'}$  quality tree for some  $q' \leq Q$  for the same quality threshold  $Q$  given assumptions 1 and 2 above.

*Proof:* The proof will proceed in two parts.

First, we show that for all  $x : x < a + F_I$  collections of refined geometry, our algorithm produces high quality acceleration structures.

We assumed that a full SAH build over each such collection will find a high quality acceleration structure. However, our algorithm does not build each  $x$ -size tree independently in one step. It may use additional geometry when choosing deeper splits in the  $x$ -size tree. It is easy to see however, that increasing the number of candidates when choosing a split plane may only decrease the SAM cost of the tree at that node. Therefore, our algorithm produces a high quality acceleration structure for all  $x$ -size subsets of the output structure.

Given that we produce a high quality tree over all  $x$ -size subsections of the geometry, each  $x$ -size section is of quality  $(cx^{1+\log_2 q})$  for some  $q : q \leq Q$  for given  $Q$  and average leaf cost  $c$ . Therefore, in our binary output tree, every  $x$ -size leaf structure (i.e. one that has true leaves as its leaves) has a SAM tree cost of  $2x^{1+\log_2 q}$ . The cost of the next  $x$ -size level up the tree is also bounded by  $cx^{1+\log_2 q} : q \leq Q$  where  $c$  is the average cost of a leaf. Since we already know the average cost of each leaf (which is an  $x$ -size tree) is  $2x^{1+\log_2 q'} : q' \leq Q$  by the same bound. The cost of our interior  $x^2$ -size tree containing its leaves is:

$$\begin{aligned} & 2x^{1+\log_2 q'} \times x^{1+\log_2 q} \\ &= 2x^{1+\log_2 q' + 1 + \log_2 q} \\ &= 2x^{2+\log_2(qq')} \\ &= 2x^{2+2\log_2 \sqrt{qq'}} \\ &= 2(x^2)^{1+\log_2 q''} \end{aligned}$$

where  $q'' = \sqrt{qq'} \leq Q$  (geometric mean). Therefore our algorithm produces high quality trees for  $x^2$ -size sub-trees. By induction to a height of  $\log_x n$  we prove that our algorithm produces a high-quality  $n$ -size output tree.  $\square$

## 5.2 Build complexity of our algorithm

We now turn our attention to the asymptotic analysis of the performance of an SAH build from an input hierarchy using our method. The analysis will require the assumption that the SAH will produce an  $O(\log a + F_I)$  depth tree over any  $\log a + F_I$  depth refinement from the input BVH hierarchy. This assumption, which will guarantee that our algorithm will produce  $O(n)$  tree nodes, is an extension of the assumption that the scene geometry has the property that the SAH will produce a  $O(\log n)$  tree over it. The extension is a local criterion over each  $a + F_I$  sized piece of it. The global version of this assumption is also made in previous work in which an

$O(n \log n)$  bound is derived [1, 14] and holds true for non-contrived scenes in practice.

*Observation 2:* Given the assumption above, building an SAH-based acceleration structure from a hierarchy is  $O(n)$  for  $n$  geometric primitives.

*Proof:* Each node in the tree considers  $x : x < a + F_I$  nodes when determining a split plane. Sorting  $O(a + F_I)$  candidates takes  $O((a + F_I) \log(a + F_I)) = C$  time. Since the completed tree has  $O(n)$  nodes by assumption, and we did  $C$  work for each node, the total work is  $O(Cn) = O(n)$  for the entire tree.  $\square$

It should be noted that any SAH build is  $\Omega(n)$  because we have to touch each object in order to add it to the tree. Therefore SAH build from hierarchy is  $\Theta(n)$ . That is, not surprisingly, an upper bound linear in the number of objects to be put into the hierarchy is tight since all objects have to be considered.

## 6 ANALYSIS OF LAZY BUILD FROM HIERARCHY

Laziness has always been a desirable property for applications that can benefit from it. In ray tracing, it can avoid large amounts of unnecessary work. In future systems with large scenes, only a small fraction of which are visible from a single viewpoint, the advantages of accelerating only the visible portion of a scene can be expected to grow. The Razor [2] system demonstrates our method of taking advantage of lazy evaluation. In this setup, it only builds visible portions of our acceleration structure at each frame. That is, it interleaves ray traversal and acceleration structure build such that only those volumes of scene space that are penetrated by a ray are built. In terms of the input BVH hierarchy, only those bounding volumes penetrated by a ray are considered when building the acceleration structure.

Before beginning the analysis of our algorithm applied to the lazy build procedure, it is important to understand the interaction between laziness and various acceleration structure build methods. Both the traditional build algorithm and the  $O(n \log n)$  [14] SAH-based build algorithm do not allow for efficient lazy implementations due to a costly  $O(n \log n)$  sort before even the first split is chosen. Scanning/binning approaches as provided in [7, 11] reduce the cost of the first split to  $O(n)$  but still touch all the geometry. In fact, any builder (even median split) that doesn't use a hierarchy over the input geometry must sift  $O(n)$  pieces of geometry after choosing the first split. In order to build a truly lazy SAH-based acceleration structure, an approach in which the initial split and sift are sub-linear in the number of nodes is necessary. Our approach does just that.

We now give an asymptotic analysis of the lazy variant of our build strategy. For a lazy system, we introduce a new variable  $v : v \leq n$ , which is the number of *visible* triangles. Laziness only provides a substantial advantage when  $v \ll n$ , so we will make that assumption here. Other assumptions of this section will differ only slightly from those in the previous section. Primarily, we assume that the SAH will produce a  $\log v$  depth tree over the set of *visible* primitives. This is essentially the same assumption as before, but now applies to each visible subset of the geometry in the scene. The implication of this assumption is that we will produce a tree with  $O(v)$  nodes.

*Observation 3:* Given the assumptions of this and the previous section, building an SAH acceleration structure is  $O(v + \log n)$ .

*Proof:* Each node in the tree considers  $x : x < a + F_I$  nodes when determining a split plane. Sorting  $O(a + F_I)$  candidates takes  $O((a + F_I) \log(a + F_I)) = C$  time. Since the completed tree is assumed to have  $O(v)$  nodes, and we did  $C$  work for each node, we do a total of  $O(Cv) = O(v)$  work for the entire tree. Additionally, we must have descended in our input hierarchy to a depth of  $\log n$  in order to access any of the geometry it contains, thus giving us a

final bound of  $O(v + \log n)$ .

$\square$

As above, it should be noted that lazy SAH build is  $\Omega(v + \log n)$  because we have to touch each visible object in order to add it to the tree, and we assume that we can only touch data through a hierarchy at depth  $\log n$ . We also have to descend to a depth of  $\log n$  in our hierarchy to extract any object. Therefore lazy SAH build from hierarchy is  $\Theta(v + \log n)$ . As a practical matter, we expect  $\log n \ll v$ , so our lazy SAH-based build is effectively linear in  $v$ .

## 7 EFFECTS OF FAST SCAN

Previous approaches, such as those provided by [7, 11] show that significant improvements in performance can be made by approximating the SAH value instead of computing it exactly. These approaches fit in nicely with algorithms that build from hierarchy. Simply, we may scan the  $a$  objects in order to approximate the SAH and find a minimum. This addition does not effect the asymptotic performance of the build algorithm (as noted above, the bound is already tight), but it does improve the constant factors. Specifically, it improves the value of  $C$ , the amount of work done at each node from  $O((a + F_I) \log(a + F_I))$  to  $O(a + F_I)$ . This improvement can be quite noticeable if you have chosen  $a$  to be large (in order to ensure high quality) or you have a hierarchy with high maximum fanout. The scan provides a convenient "performance safety-net" in case assumptions about fanout are broken in a real-world example. Our experimental results show that the effects of the fast scan approximation combine nicely with both laziness and the use of hierarchy.

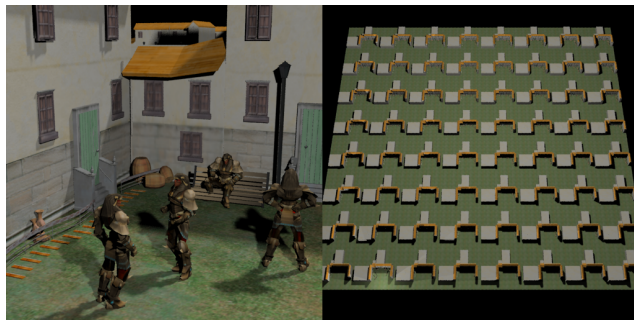


Figure 3: Scene (Courtyard.64) used for gathering experimental results. In the near viewpoint shown at left, 9392 primitives are visible. In the far viewpoint shown at right, 89040 primitives are visible. It has 541023 primitives total.



Figure 4: Scene (Conference Room) used for gathering experimental results. In the near viewpoint shown at left, 13031 primitives are visible. In the far viewpoint shown at right, 156437 primitives are visible. It has 494706 primitives total.

			crtyrd near (n=541023) (v=9392)	crtyrd near	crtyrd far (n=541023) (v=89040)	crtyrd far	conf near (n=494706) (v=13031)	conf near	conf far (n=494706) (v=156437)	conf far
Lazy	Scan	Hier	build time	trace time	build time	trace time	build time	trace time	build time	trace time
+	+	+	0.116 s	1.302 s	1.608 s	1.388 s	0.129 s	1.631 s	0.674 s	2.150 s
+	+	-	0.896 s	1.305 s	1.873 s	1.399 s	0.825 s	2.464 s	1.208 s	3.529 s
+	-	+	0.401 s	1.287 s	3.985 s	1.361 s	0.403 s	1.577 s	2.095 s	2.031 s
+	-	-	3.214 s	1.285 s	9.651 s	1.366 s	3.397 s	1.673 s	5.898 s	2.099 s
-	+	+	3.843 s	1.303 s	3.956 s	1.389 s	1.190 s	1.629 s	1.378 s	2.152 s
-	+	-	4.089 s	1.298 s	4.205 s	1.397 s	1.403 s	2.465 s	1.650 s	3.507 s
-	-	+	6.620 s	1.290 s	6.734 s	1.369 s	5.507 s	1.580 s	5.668 s	2.032 s
-	-	-	12.270 s	1.283 s	12.385 s	1.368 s	10.792 s	1.673 s	10.995 s	2.096 s

Table 1: Build times for a k-d tree with various fast-build capabilities enabled and disabled. Results are presented for two viewpoints of each scene, Courtyard.64, which has  $n=541023$  total primitives and Conference which has  $n=494706$  total primitives. From the first viewpoint ( $v=9392$ ) fewer primitives are visible than from the second ( $v=89040$ ). Results are also presented for two viewpoints of the conference scene, which has  $n=494706$ . These results were taken from Razor on a single core of an Intel Core 2 Duo 2.667 GHz machine. Build time includes some related activities in Razor’s geometry subsystem as well as the kd-tree build itself. Razor’s design is optimized to use a separate acceleration structure for polygons within a subdivision patch, but for these measurements each of these separate acceleration structures was restricted to consist of single quad so as to focus on kd-tree build time. Thus, these results are not indicative of the build performance per triangle that Razor achieves when appropriately configured.

## 8 EXPERIMENTAL RESULTS

In this section we present experimental results obtained from the Razor implementation and demonstrate that the techniques described above improve build performance and that the measured asymptotic behavior approximately matches that predicted by the theoretical results. It should be noted that the Razor acceleration structure build can be slower than other builders. This has to do with the complexity of the multi-resolution tree constructed and the associated overhead. This overhead does not affect the relative performance improvements provided by our algorithm. It also causes builds to be slightly faster for the smaller visible scene even though the lazy part of the build system is off.

For these results, we use a “Courtyard (64)” scene that has a scene graph hierarchy, but not a particularly good one; for example the fanout at some of the leaf nodes is 1568. Any reasonable scene graph should provide hierarchy as good or better than this one.

Table 1 shows that build from hierarchy, lazy build, and scan-based SAH build each contribute to improved build performance. In particular, the combination of lazy build with build from hierarchy is much more effective than either alone when the number of visible primitives is much smaller than the total number of primitives (i.e.  $v \ll n$ ). For the  $v = 9392$  view, there is a 7.7x speedup when using lazy+scan+hierarchy as compared to just lazy+scan. These results experimentally confirm the theoretical result that a  $O(v + \log n)$  build is significantly faster than a  $O(n)$  build.

In contrast, the difference between  $O(n)$  build from hierarchy+scan and  $O(n \log n)$  build from scan is less significant. This smaller difference is due to the slow growth of  $\log n$ . Thus, the most important practical implication of our analysis is that it is important to use lazy build *and* build from hierarchy, rather than either technique in isolation.

Table 1 also shows how traversal performance varies as our various techniques are applied. There is essentially no difference in traversal performance when build from hierarchy is used instead of build from polygon soup for the courtyard scene. Scan-based build slightly reduces trace performance as compared to a sort-based build, but never by more than 2% in that scene. In the conference scene, hierarchy provides a tracing speed improvement in all configurations. This is because a good hierarchy can overcome the short-sightedness of the greedy SAH to produce better (more informed) trees. Most noticeable is the fact that with the near camera position, the hierarchy based builder produces a tree that is  $> 5\%$  faster to trace than the greedy full SAH. In this scene, scan produces

noticeably poor trees without hierarchy. This is due to the scan only considering one axis for its top level splits.

Figure 5 shows how build time grows with increasing  $n$  when build from hierarchy is used, without lazy build, but with fast scan. The earlier theoretical results predict  $O(n)$  behavior under this configuration, and the experimental results match this predicted behavior almost perfectly.

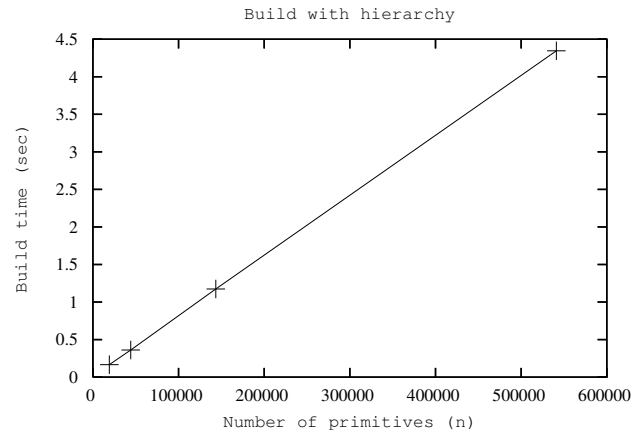


Figure 5: Performance of build-from-hierarchy as a function of  $n$ , the number of primitives. Lazy is disabled, and scan is enabled. We vary  $n$  by adding additional occluded geometry to the scene.

Figure 6 shows how build time grows with increasing  $v$  when build from hierarchy is combined with lazy build and fast scan. The earlier theoretical results predict  $O(v)$  behavior under this configuration. The experimental results roughly match this predicted behavior but there are clearly some other effects present as well. We note that it is difficult to measure the effect of varying  $v$  precisely, because it is not possible to change  $v$  without also indirectly changing other variables in the system. In this case, we varied  $v$  by choosing different viewpoints within the same model, but different viewpoints do not behave exactly the same way even if they have the same amount of visible geometry. We also suspect that memory hierarchy effects are causing non-linear behavior in some regions of the curve. Despite these issues, it is clear that build time grows

approximately linearly with  $v$  over large regions of the curve, as expected.

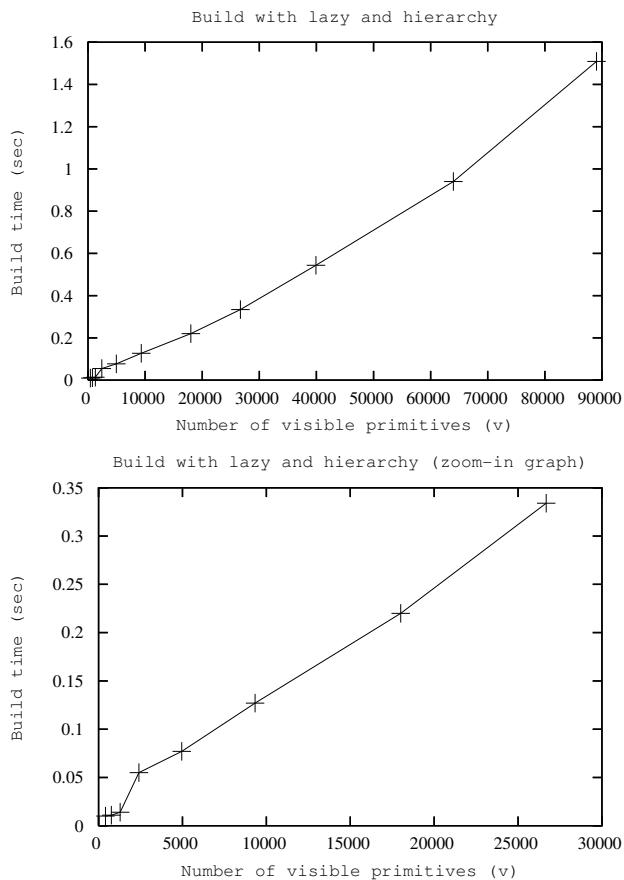


Figure 6: Performance of lazy build from hierarchy as a function of  $v$ , the number of visible primitives. Top: full graph, Bottom: zoom-in graph for small  $v$ .

## 9 ADDITIONAL CONSIDERATIONS

Additional adaptations can be applied to our build from hierarchy scheme to improve the quality of the SAH structure produced. None of these quality optimizations affect the asymptotic runtime of the algorithm, but they can improve the quality of output structures and avoid pitfalls in case assumptions about the scene fail.

The first optimization involves allowing the tree to access additional hierarchy nodes beyond the initial  $O(a + F_I)$  when the SAH cannot find a viable split among the first  $O(a + F_I)$  candidates. This can allow the builder to dig further in an attempt to find a split instead of aborting early and producing a large/low-quality leaf node. If the allowance is any constant factor of  $(a + F_I)$ , the asymptotic analysis gives identical bounds. Any adaptive scheme must assume some amortized bound on the number of occurrences of this allowance. For example, in our system on courtyard, the number of times more than 500 nodes are required to make a split is negligible (always 0). The number of times more than 30 nodes are required is also extremely small ( $< 100$ ).

The second optimization is the allowance of additional nodes for the sake of split plane evaluation in a scan-based builder. As described in [7], the SAH may be approximated by sampling the SAH function in discrete locations. We may allow the algorithm to refine a constant number  $d$  of nodes that straddle some sample. This will

improve the quality of the scan approximation by allowing for detailed evaluation of the sample points without refining nodes that are clearly between samples and would not provide any additional information to the scanner. This also only modestly affects constant factors in the algorithm, simply changing  $C$  from  $O(a + F_I)$  to  $O(a + F_I + d)$ .

Finally, as we noted above, this scheme can be applied to the construction any SAH-based acceleration structure, in particular a BVH. If we are building a BVH acceleration hierarchy, we can exploit temporal coherence by using the previous frame’s BVH as the hierarchy for the next frame. Since most of the split planes will be very similar or identical (we already have a high-quality input hierarchy), we could pick  $a$  to be very small and thus provide extremely good constant factors for the build algorithm. With the addition of lazy evaluation, any part of the tree that is not rebuilt in the previous frame would simply be updated using the simple BVH update algorithm. That part of the tree would not be of the same quality as the rebuilt portion, but it would not matter much because it would only be used as a starting hierarchy for a future construction. Given the fixed number of nodes in a BVH and other nice properties, this strategy could perform very well.

## 10 CONCLUSIONS

We have introduced a scheme that takes advantage of the a priori structure of a scene to improve the construction speed of ray tracing acceleration hierarchies over previous approaches. In particular, for building acceleration hierarchies using the SAH, we show an  $O(n)$  performance bound given reasonable assumptions on scene structure that we have found to hold in practice for non-pathological scenes. This compares favorably to the bound of  $O(n \log n)$  for building such structures from scratch and matches the trivial lower bound for any scheme that is required to touch all  $n$  input elements. We have also shown how our scheme can be combined with other strategies such as lazy evaluation and the SAH approximation schemes of [7, 11] to improve the runtime performance even further and shown that in conjunction with lazy evaluation, asymptotic runtime drops to  $O(v)$ . An important result is that the combination of build from hierarchy with lazy build is asymptotically faster than either method alone. Our scheme is not only asymptotically fast, but it is robust and quite practical to use in a real ray-tracing context. Indeed, the k-d tree building scheme with lazy evaluation and the scan-based SAH approximation of [7] is used in our Razor system [2] to give improvements of an order of magnitude in speed over building these trees from scratch, producing very high-quality acceleration structures for dynamic scenes at interactive rates.

## ACKNOWLEDGEMENTS

The authors wish to thank the rest of the Razor development team for their help and discussions: Peter Djeu, Gordon Stoll, Rui Wang and Ikrima Elhassan. This work was supported in part by NSF CAREER award #0546236, by a research grant from Intel Corporation, and by a fellowship from the IC<sup>2</sup> Institute at the University of Texas at Austin. The authors would especially like to thank Jim Hurley at Intel for his enthusiastic support of real-time ray tracing research.

## REFERENCES

- [1] C. Benthin, I. Wald, M. Scherbaum, and H. Friedrich. Ray tracing on the CELL processor. In *Proceedings of the 2006 IEEE Symposium on Interactive Ray Tracing*, pages 15–23, 2006.
- [2] P. Djeu, W. Hunt, R. Wang, I. Elhassan, G. Stoll, and W. R. Mark. Razor: An architecture for dynamic multiresolution ray tracing. Technical report, University of Texas at Austin Dep. of Comp. Science, 2007. Conditionally accepted to ACM Transactions on Graphics.
- [3] J. Goldsmith and J. Salmon. Automatic creation of object hierarchies for ray tracing. *IEEE Computer Graphics and Applications*, 7(5):14–20, 1987.

- [4] J. Günther, H. Friedrich, H.-P. Seidel, and P. Slusallek. Interactive ray tracing of skinned animations. *The Visual Computer*, 22(9):785–792, Sept. 2006. (Proceedings of Pacific Graphics 2006).
- [5] V. Havran. PhD thesis.
- [6] V. Havran and J. Bittner. On improving KD tree for ray shooting. In *Proceedings of WSCG*, pages 209–216, 2002.
- [7] W. Hunt, G. Stoll, and W. Mark. Fast kd-tree construction with an adaptive error-bounded heuristic. In *Proceedings of the 2006 IEEE Symposium on Interactive Ray Tracing*, 2006.
- [8] C. Lauterbach, S.-E. Yoon, D. Tuft, and D. Manocha. RT-DEFORM: Interactive ray tracing of dynamic scenes using BVHs. In *Proceedings of the 2006 IEEE Symposium on Interactive Ray Tracing*, pages 39–45, 2006.
- [9] J. D. MacDonald and K. S. Booth. Heuristics for ray tracing using space subdivision. In *Proceedings of Graphics Interface*, pages 152–63, 1989.
- [10] M. Pharr and G. Humphreys. *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann, 2004.
- [11] S. Popov, J. Günther, H.-P. Seidel, and P. Slusallek. Experiences with streaming construction of SAH kd-trees. In *Proceedings of the 2006 IEEE Symposium on Interactive Ray Tracing*, 2006.
- [12] C. Wächter and A. Keller. Instant ray tracing: The bounding interval hierarchy. In *Proceedings of the 17th Eurographics Symposium on Rendering*, pages 139–149, 2006.
- [13] I. Wald. On fast construction of SAH-based bounding volume hierarchies. In *Proceedings of the 2007 IEEE/EG Symposium on Interactive Ray Tracing*. IEEE, 2007. (to appear).
- [14] I. Wald and V. Havran. On building fast kd-trees for ray tracing, and on doing that in  $O(N \log N)$ . In *Proceedings of the 2006 IEEE Symposium on Interactive Ray Tracing*, pages 61–70, 2006.
- [15] I. Wald, T. Ize, A. Kensler, A. Knoll, and S. G. Parker. Ray tracing animated scenes using coherent grid traversal. *ACM Transactions on Graphics*, 25(3):485–493, 2006. (Proceedings of ACM SIGGRAPH).
- [16] I. Wald, W. R. Mark, J. Günther, S. Boulos, T. Ize, W. Hunt, S. G. Parker, and P. Shirley. State of the art in ray tracing animated scenes. In *Eurographics 2007 State of the Art Reports*. Eurographics Association, 2007.
- [17] S.-E. Yoon, S. Curtis, and D. Manocha. Ray tracing dynamic scenes using selective restructuring. In *Proceedings of the 18th Eurographics Symposium on Rendering*, pages 73–84. Eurographics Association, 2007.